

# **USB HID for Linux USB**

**Brad Hards**  
**Sigma Bravo Pty Ltd.**



## **USB HID for Linux USB**

by Brad Hards

This document is an early draft of a guide to using USB Human Interface Devices with the GNU/Linux operating system. It describes how Human Interface Device (HID) class devices are handled, and how to program them, primarily focussing on user-space interfaces.

This document is matched to Linux kernel revision 2.4.19-pre7. Some of this information may not work in exactly the same way on earlier or kernel revisions. Known differences to the 2.5 series kernels are also included where appropriate.



# Table of Contents

<b>I. How Linux handles USB HID devices</b> .....	7
1. Introduction .....	7
The Universal Serial Bus.....	7
Host Controllers.....	7
USB Devices and Transfer Characteristics .....	7
USB Device Drivers .....	9
2. Configuring Linux HID support.....	11
USB subsystem.....	11
Input subsystem.....	11
3. The HID device interface .....	13
What is the hiddev interface .....	13
How HID devices relate to the hiddev interface.....	13
Getting the version of the hiddev interface .....	13
Getting information about the HID device.....	14
Determining which Applications a device uses.....	15
Reading from the HID device interface.....	16
4. The event interface .....	17
How HID relates to the event interface.....	17
Getting the version of the evdev interface.....	17
Getting information about the HID device identity .....	18
Determining the device capabilities and features.....	19
Getting user input from the device .....	24
Sending information to the device .....	25
Modifying key repeat settings .....	25
5. Other approaches to Linux HID support.....	27
The keyboard interface .....	27
The mouse interface .....	27
The joystick interface.....	27
A new kernel interface .....	27
<b>II. Examples</b> .....	29
6. HIDDEV examples.....	29
7. EVDEV examples .....	35
<b>III. About this Guide</b> .....	59
8. Contributors .....	59
9. Availability and licensing.....	61
Obtaining updates and translations.....	61
License.....	61
10. Corrections .....	63



# Chapter 1. Introduction

## The Universal Serial Bus

In 1994 an alliance of four industrial partners (Compaq, Intel, Microsoft and NEC) started to specify the Universal Serial Bus (USB). The bus was originally designed with these intentions:

- Connection of the PC to the telephone
- Ease-of-use
- Port expansion

The specification (version 1.0) was first released in January 1996, with some changes being issued as version 1.1 in September 1998. The latest version of the specification is Version 2.0, which was release in FIXME.

The USB is strictly hierarchical and it is controlled by one host. The host uses a master / slave protocol to communicate with attached USB devices. This means that every kind of communication is initiated by the host and devices cannot establish any direct connection to other devices. This seems to be a drawback in comparison to other bus architectures but it is not because the USB was designed as a compromise of costs and performance. The master / slave protocol solves implicitly problems like collision avoidance or distributed bus arbitration. The current implementation of the USB allows 127 devices to be connected at the same time.

## Host Controllers

Most modern motherboard chipsets provide a USB host controller. Older machines which are not equipped with a USB host controller can be upgraded using a PCI cards with a host controller built in.

There are a range of different host controller designs, however the three most common of designs are supported under Linux - Open Host Controller Interface (OHCI), Universal Host Controller Interface (UHCI) and Enhanced Host Controller Interface (EHCI).

## USB Devices and Transfer Characteristics

There are a wide range of USB devices intended for a wide range of purposes, and this means that implementation details can vary widely.

A device can be self powered, bus powered or both. The USB can provide a power supply up to 500mA for its devices. If there are only bus powered devices on the bus the maximum power dissipation could be exceeded and therefore self powered devices exist. They need to have their own power supply. Devices that support both power types can switch to self powered mode when attaching an external power supply.

Even the maximum communication speed can differ for particular USB devices. The USB specification differentiates between low speed and full speed devices. Low speed devices (such as mice, keyboards, joysticks etc.) communicate at 1.5MBit/s and have only limited capabilities. Full speed devices (such as audio and video systems) can use up to 90% of the 12Mbit/s which is about 10Mbit/s including the protocol overhead. High speed devices (introduced in USB 2.0) can theoretically get to around 400Mbps, although a lot of other factors come into play at this speed.

## Hubs

Physically there exist one, two or four USB ports at the rear, front or side panel of a computer. These ports can be used to attach normal devices or a hub. A hub is a USB device which extends the number of ports to connect other USB devices. The maximum number of user devices is reduced by the number of hubs on the bus (i.e. if you attach 50 hubs, then at most 77 (=127-50) additional devices can be attached. Hubs are always full speed (for USB 1.1 compliant hubs) or high speed (for USB 2.0 compliant hubs) devices. If the hub is self powered, then any device can be attached to it. However if the hub is bus powered, then only low power (100mA max) devices can be attached to it. A bus powered hub should not be connected to another bus powered hub - you should alternate between bus powered and self powered hubs.

Normally the physical ports of the host controller are handled by a virtual root hub. This hub is simulated by the host controllers device driver and helps to unify the bus topology. So every port can be handled in the same way by the USB subsystem's hub driver.

## Data Flow Types

The communication on the USB is done in two directions and uses four different transfer types. Data directed from the host to a device is called downstream or OUT transfer. The other direction is called upstream or IN transfer. Depending on the device type different transfer variants are used:

- *Control transfers* are used to request and send reliable short data packets. It is used to configure devices and every one is required to support a minimum set of control commands. The standard commands are:

```
GET_STATUS
CLEAR_FEATURE
SET_FEATURE
SET_ADDRESS
GET_DESCRIPTOR
SET_DESCRIPTOR
GET_CONFIGURATION
SET_CONFIGURATION
GET_INTERFACE
SET_INTERFACE
SYNCH_FRAME
```

Further control commands can be used to transfer vendor specific data.

- *Bulk transfers* are used to request or send reliable data packets up to the full bus bandwidth. Devices like scanners or scsi adapters use this transfer type.

- *Interrupt transfers* are similar to bulk transfers which are polled periodically. If an interrupt transfer was submitted the host controller driver will automatically repeat this request in a specified interval (1ms - 127ms).
- *Isochronous transfers* send or receive data streams in realtime with guaranteed bus bandwidth but without any reliability. In general these transfer types are used for audio and video devices.

## Human Interface Devices (HID)

The HID class consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of HID class devices include:

Keyboards and pointing devices for example, standard mouse devices, trackballs, and joysticks.

Front-panel controls for example: knobs, switches, buttons, and sliders.

Controls that might be found on devices such as telephones, VCR remote controls, games or simulation devices.

## USB Device Drivers

Finding device drivers for USB devices presents some interesting situations. In some cases the whole USB device is handled by a single device driver. In other cases, each interface of the device has a separate device driver.



## Chapter 2. Configuring Linux HID support

This chapter provides a brief overview of which kernel options are relevant for Linux HID support.

This chapter is currently just a stub for more information that will be provided in a later version.

### USB subsystem

Refer to the Linux USB Guide for more information on configuring USB. If you are trying to use some of the examples in the hiddev chapter, you should turn the HID input layer support(`CONFIG_USB_HIDINPUT`) off, and turn the `/dev/hiddev` raw HID device support(`CONFIG_USB_HIDDEV`) on.

If you are trying to use some of the examples in the event chapter, you must turn the HID input layer support(`CONFIG_USB_HIDINPUT`) on. You can have the `/dev/hiddev` raw HID device support(`CONFIG_USB_HIDDEV`) on or off - it won't make any difference.

### Input subsystem

You should turn all options on.



## Chapter 3. The HID device interface

This chapter provides information on using the hiddev interface.

### What is the hiddev interface

The hiddev interface provides access to certain HID devices at a relatively low level. It can be considered a "raw" interface to HID functionality from userspace.

### How HID devices relate to the hiddev interface

As stated in a prior chapter, the USB host controller interacts with the USB devices on its bus. The USB core functionality ensures that information associated with HID devices is delivered to the HID device driver (`hid.o`). The HID device driver then uses the Application of the particular USB device interface to determine whether the device is an input type device (such as a keyboard, mouse, joystick or gamepad) or is some other HID device (such as a USB monitor control, or a uninterruptable power supply).

If the device appears to be an input type device, and the HID device driver was compiled to support the input subsystem, then the information from the HID device will be provided to the input subsystem (where it can be accessed using joystick, mouse, keyboard or event interfaces, as discussed in later chapters).

If the device does not appear to be an input device (or there is no input subsystem support in the HID device driver) and the HID device driver has been compiled to support the hiddev driver, then the information provided by the HID device will be made available on the hiddev interface, which is the subject of this chapter.

### Getting the version of the hiddev interface

The hiddev interface supports determining the version of the hiddev code, using the `HIDIOCGVERSION` `ioctl`. The argument is an `int`, and it is meant to be interpreted as a major version (two high bytes), a minor version (third byte) and a patch level (low byte).

Lets look at an example of the `HIDIOCGVERSION` `ioctl` call.

**About example code:** This example, and several others in this document, are not complete, nor are they meant to show good programming style. Instead, they are meant to be illustrative of a particular feature. Real programs should, for example, check return values for all functions that can potentially fail.

Complete examples (that will compile with `gcc -Wall -W`) are provided in the second part of this document.

#### Example 3-1. HIDIOCGVERSION example

```
/* ioctl() accesses the underlying driver */
ioctl(fd, HIDIOCGVERSION, &version);
```

```
/* the HIDIOCGVERSION ioctl() returns an int */
/* so we unpack it and display it */
printf("hiddev driver version is %d.%d.%d\n",
       version >> 16, (version >> 8) & 0xff, version & 0xff);
```

The example should be relatively obvious. The first argument is an open file descriptor for the hiddev device node (for example, /dev/usb/hiddev0). Also note that you have to pass a pointer to the integer variable, not the variable itself, as the third argument to the `ioctl` call.

## Getting information about the HID device

The hiddev interface supports getting information associated with the device using the `HIDIOCGDEVINFO` `ioctl`. The argument is a pointer to a `hiddev_devinfo` structure.

The `hiddev_devinfo` structure is defined as:

```
struct hiddev_devinfo {
    unsigned int bustype;
    unsigned int busnum;
    unsigned int devnum;
    unsigned int ifnum;
    short vendor;
    short product;
    short version;
    unsigned num_applications;
};
```

Lets look at an example of the `HIDIOCDEVINFO` `ioctl` call.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -w`) is provided in the second part of this document.

### Example 3-2. HIDIOCDEVINFO example

```
/* suck out some device information */
ioctl(fd, HIDIOCGDEVINFO, &device_info);

printf("vendor 0x%04hx product 0x%04hx version 0x%04hx ",
       device_info.vendor, device_info.product, device_info.version);
printf("has %i application%s ", device_info.num_applications,
       (device_info.num_applications==1?"":"s"));
printf("and is on bus: %d devnum: %d ifnum: %d\n",
       device_info.busnum, device_info.devnum, device_info.ifnum);
```

The `vendor`, `product` and `version` fields correspond to the vendor identification, product identification and BCD revision of the device.

The *busnum*, *devnum* and *ifnum* fields correspond to the logical location of the device on the USB bus.

The *num\_applications* field is the number of applications in the HID descriptor.

## Determining which Applications a device uses

Refer to the HID specification<sup>1</sup> for more details on what an application is, and refer to HID Usage Tables<sup>2</sup>, HID Usage Tables for Power Devices<sup>3</sup>, Monitor Class Specification<sup>4</sup> or HID Point of Sale Usage Tables<sup>5</sup> for the valid applications.

You can determine the application (or applications) that the device supports using the `HIDIOCAPPLICATION` `ioctl` call.

Lets look at an example of the `HIDIOCDEVINFO` `ioctl` call.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -w`) is provided in the second part of this document.

### Example 3-3. HIDIOCDEVINFO example

```
ioctl(fd, HIDIOCGDEVINFO, &device_info);

/* Now that we have the number of applications (in the
 * device_info.num_applications field),
 * we can retrieve them using the HIDIOCAPPLICATION ioctl()
 * applications are indexed from 0..{num_applications-1}
 */
for (yalv = 0; yalv < device_info.num_applications; yalv++) {
    appl = ioctl(fd, HIDIOCAPPLICATION, yalv);
    if (appl > 0) {
printf("Application %i is 0x%x ", yalv, appl);
/* The magic values come from various usage table specs */
switch ( appl >> 16)
    {
        case 0x01 :
printf("(Generic Desktop Page)\n");
break;
        case 0x0c :
printf("(Consumer Product Page)\n");
break;
        case 0x80 :
printf("(USB Monitor Page)\n");
break;
        case 0x81 :
printf("(USB Enumerated Values Page)\n");
break;
        case 0x82 :
printf("(VESA Virtual Controls Page)\n");
break;
        case 0x83 :
printf("(Reserved Monitor Page)\n");

```

```
break;
    case 0x84 :
printf("(Power Device Page)\n");
break;
    case 0x85 :
printf("(Battery System Page)\n");
break;
    case 0x86 :
    case 0x87 :
printf("(Reserved Power Device Page)\n");
break;
    default :
printf("(Unknown page - needs to be added)\n");
}
}
```

This example uses the `HIDIOCDEVINFO ioctl1` that we saw in the previous example, which allows us to determine how many applications the device has. The actual applications are the return value from the `ioctl` call. To determine all applications, the third argument is iterated from 0 to the total number applications, less one (since it is zero based).

## Reading from the HID device interface

Not yet completed.

## Notes

1. [http://www.usb.org/developers/data/devclass/HID1\\_11.pdf](http://www.usb.org/developers/data/devclass/HID1_11.pdf)
2. [http://www.usb.org/developers/data/devclass/Hut1\\_11.pdf](http://www.usb.org/developers/data/devclass/Hut1_11.pdf)
3. <http://www.usb.org/developers/data/devclass/pdcv10.pdf>
4. <http://www.usb.org/developers/data/devclass/usbmon10.pdf>
5. [http://www.usb.org/developers/data/devclass/pos1\\_02.pdf](http://www.usb.org/developers/data/devclass/pos1_02.pdf)

## Chapter 4. The event interface

This chapter provides information on using the event interface. While the examples are based on USB devices, this information is actually applicable to just about any input device.

### How HID relates to the event interface

There is a character event device created for each USB HID interface (that has an application field that indicates it is an input device), and potentially for other input devices (if any) that are attached to the system. Note that a single device may have more than one HID interface (for example, some "multimedia" keyboards have the additional buttons on a separate interface, since they can't be supported by the boot protocol) and hence may have more than one event device.

This event interface is in addition to other, legacy, character devices that may also be created (for example, normal keyboard input, `/dev/input/mouse0` and `/dev/input/mice`, or `/dev/input/js0`). Refer to the next Chapter for options for these legacy character devices.

### Getting the version of the evdev interface

The evdev interface supports determining the version of the hiddev code, using the `EVIIOCGVERSION` `ioctl`. The argument is an `int`, and it is meant to be interpreted as a major version (two high bytes), a minor version (third byte) and a patch level (low byte).

Lets look at an example of the `EVIIOCGVERSION` `ioctl` call.

**About example code:** This example, and several others in this document, are not complete, nor are they meant to show good programming style. Instead, they are meant to be illustrative of a particular feature. Real programs should, for example, check return values for all functions that can potentially fail.

Complete examples (that will compile with `gcc -Wall -W`) are provided in the second part of this document.

#### Example 4-1. EVIIOCGVERSION example

```
ioctl(fd, EVIIOCGVERSION, &version);

/* the EVIIOCGVERSION ioctl() returns an int */
/* so we unpack it and display it */
printf("evdev driver version is %d.%d.%d\n",
       version >> 16, (version >> 8) & 0xff, version & 0xff);
```

The example should be relatively obvious. The first argument is an open file descriptor for the hiddev device node (for example, `/dev/input/evdev0`). Also note that you have to pass a pointer to the integer variable, not the variable itself, as the third argument to the `ioctl` call.

The more observant readers would have noted the similarity between this `ioctl` and the `HIDIOCGVERSION` `ioctl` in the previous chapter. The format is the same, and the result is in the same form.

## Getting information about the HID device identity

The `evdev` interface supports getting information associated with the device using the `EVIIOCGID` `ioctl`. The argument is a pointer to an `input_devinfo` structure.

The `input_devinfo` structure is defined as:

```
struct input_devinfo {
    uint16_t bustype;
    uint16_t vendor;
    uint16_t product;
    uint16_t version;
};
```

The `bustype` field is always `BUS_USB` for USB devices, but has other values for other devices. In the 2.5.x and later series kernels, the input subsystem is used for all input devices, and this value can take a wide range of values. In the 2.4.x series kernels, the input subsystem has restricted application (for example, USB and Apple Desktop Bus).

The `vendor`, `product` and `version` elements are associated with the device. For a USB device, they have one-to-one correspondence with `wVendor`, `wProduct` and `bcdRevision` fields in the USB device descriptor.

Lets look at an example of the `EVIIOCGID` `ioctl` call.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -w`) is provided in the second part of this document.

### Example 4-2. EVIIOCGID example

```
ioctl(fd, EVIIOCGID, &device_info);

printf("vendor 0x%04hx product 0x%04hx version 0x%04hx is on",
       device_info.vendor, device_info.product,
       device_info.version);
switch ( device_info.bustype)
{
    case BUS_USB :
        printf(" a Universal Serial Bus\n");
        break;
    default:
        printf(" an unknown bus type: 0x%04hx", device_info.bustype);
}
```

Another useful `ioctl` for getting information on the device is `EVIIOCGNAME`. It returns a char array, containing the name of the device.

Lets look at an example of the `EVIIOCGNAME` `ioctl` call.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -W`) is provided in the second part of this document.

### Example 4-3. `EVIIOCGNAME` example

```
char name[256]= "Unknown";

if(ioctl(fd, EVIIOCGNAME(sizeof(name)), name) < 0) {
    perror("evdev ioctl");
}

printf("The device on %s says it's name is %s\n", argv[1], name);
```

The `EVIIOCGNAME` `ioctl` returns the length of the string if it succeeds, and a negative number if it fails. For USB devices, the string is made by concatenating the strings from *iManufacturer* and *iProduct* string descriptors reported by the USB device. Note that these are both optional.

If the string is too long to fit into the argument, it will be truncated.

If it seems strange that the third argument isn't `&name`, remember that the name of an array is the same as a pointer to the first element. Therefore `&name` would be a pointer to a pointer to the first element, which is not what we want. If you really want to use a dereference, use `&(name[0])`.

On my system, with a joystick, running a version of this code produces:

```
[root@localhost evdev-example3]# ./example3 /dev/input/event0
The device on /dev/input/event0 says it's name is Microsoft SideWinder
Joystick
```

## Determining the device capabilities and features

One of the major advantages of HID is that devices are *self-describing*. You can tell what sort of capabilities and features the device has in a standardised way. This section describes how to determine what the capabilities and features of a particular HID device are.

The types of features that are supported by the event interface are:

- `EV_KEY` which is used for absolute binary results, such as keys and buttons.
- `EV_REL` which is used for relative results, such as the axes on a mouse.
- `EV_ABS` which is used for absolute integer results, such as the axes on a joystick.
- `EV_MSC` which is reserved for future miscellaneous uses.

- EV\_LED which is used for LEDs and similar indications.
- EV\_SND which is used for sound output, such as buzzers.
- EV\_REP which is used for key repeat effects.
- EV\_FF which is used for force feedback effects.

Within each feature type, there can be a wide range of features. For example, the EV\_REL feature type distinguishes between X, Y and Z axes and horizontal and vertical wheels. Similarly, the EV\_KEY feature type distinguishes between literally hundreds of different keys and buttons.

These capabilities (or features, if you prefer) can be determined through the event interface, using the EVIOCGBIT `ioctl`. This function allows you to determine the types of features that any particular device supports (for example, whether it has keys or buttons, or not). It further allows you to determine the specific features that are supported (for example, which keys or buttons are present).

The EVIOCGBIT `ioctl` takes four arguments. If we consider it as

```
ioctl(fd,  
EVIOCGBIT(ev_type, max_bytes), bitfield)
```

, then the `fd` argument is an open file descriptor, `ev_type` is the type of features to return (with 0 as a special case, indicating that the list of all features types that are supported should be returned, rather than the list of particular features for that type), `max_bytes` shows the upper limit on how many bytes should be returned, and `bitfield` is a pointer to the memory area where the result should be copied. The return value is the number of bytes actually copied (on success) or a negative error code (on failure).

Lets look at a couple of examples of the EVIOCGBIT `ioctl` call. The first example shows how to determine the types of features that are present.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -w`) is provided in the second part of this document.

#### Example 4-4. EVIOCGBIT event list example

```
/* this macro is used to tell if "bit" is set in "array"  
 * it selects a byte from the array, and does a boolean AND  
 * operation with a byte that only has the relevant bit set.  
 * eg. to check for the 12th bit, we do (array[1] & 1<<4)  
 */  
#define test_bit(bit, array)    (array[bit/8] & (1<<(bit%8)))  
...  
uint8_t evtype_bitmask[EV_MAX/8 + 1];  
  
ioctl(fd, EVIOCGBIT(0, sizeof(evtype_bitmask)), evtype_bitmask) < 0)  
  
printf("Supported event types:\n");
```

```

for (yalv = 0; yalv < EV_MAX; yalv++) {
    if (test_bit(yalv, evtype_bitmask)) {
        /* this means that the bit is set in the event types list */
        printf(" Event type 0x%02x ", yalv);
        switch ( yalv)
        {
            case EV_KEY :
                printf(" (Keys or Buttons)\n");
                break;
            case EV_ABS :
                printf(" (Absolute Axes)\n");
                break;
            case EV_LED :
                printf(" (LEDs)\n");
                break;
            case EV_REP :
                printf(" (Repeat)\n");
                break;
            default:
                printf(" (Unknown event type: 0x%04hx)\n", yalv);
        }
    }
}

```

The result of running the complete form of this code with a USB joystick on my system are:

```

[root@localhost evdev-example4]# ./example4a /dev/input/event0
Supported event types:
Event type 0x01 (Keys or Buttons)
Event type 0x03 (Absolute Axes)

```

The same code with a USB keyboard shows:

```

[root@localhost evdev-example4]# ./example4a /dev/input/event1
Supported event types:
Event type 0x01 (Keys or Buttons)
Event type 0x11 (LEDs)
Event type 0x14 (Repeat)

```

The keyboard happens to have a second interface (for the “multimedia” keys). This second interface shows:

```

[root@localhost evdev-example4]# ./example4a /dev/input/event2
Supported event types:
Event type 0x01 (Keys or Buttons)

```

Note that the previous example of `EVIOCGBIT ioctl` showed how to determine which function types are supported. Let’s consider an example that shows how to determine the particular features that are available within each feature type. The example that follows looks at what LEDs are supported by a particular device within the `EV_LED` feature type.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -w`) is provided in the second part of this document.

**Example 4-5. EVIOCGBIT LED example**

```

uint8_t led_bitmask[LED_MAX/8 + 1];
...
memset(led_bitmask, 0, sizeof(led_bitmask));
ioctl(fd, EVIOCGBIT(EV_LED, sizeof(led_bitmask)), led_bitmask) < 0)

printf("Supported LEDs:\n");
for (yalv = 0; yalv < LED_MAX; yalv++) {
    if (test_bit(yalv, led_bitmask)) {
        /* this means that the bit is set in the event types list */
        printf(" LED 0x%02x ", yalv);
        switch (yalv)
        {
            case LED_NUML :
                printf(" (Num Lock)\n");
                break;
            case LED_CAPSL :
                printf(" (Caps Lock)\n");
                break;
            case LED_SCROLLL :
                printf(" (Scroll Lock)\n");
                break;
            case LED_COMPOSE :
                printf(" (Compose)\n");
                break;
            case LED_KANA :
                printf(" (Kana)\n");
                break;
            case LED_SLEEP :
                printf(" (Sleep)\n");
                break;
            case LED_SUSPEND :
                printf(" (Suspend)\n");
                break;
            case LED_MUTE :
                printf(" (Mute)\n");
                break;
            case LED_MISC :
                printf(" (Miscellaneous)\n");
                break;
            default:
                printf(" (Unknown LED type: 0x%04hx)\n", yalv);
        }
    }
}

```

Note that the `ioctl` is basically the same, except that the second argument has changed from 0 to `EV_LED`, and that the length of the bitfield is now determined by `LED_MAX`, rather than `EV_MAX`. Naturally, the meaning of the bits in the bitfield has changed.

Running the complete form of this program on a typical keyboard will produce:

```

[root@localhost evdev-example4]# ./example4b /dev/input/event1
Supported LEDs:
LED type 0x00 (Num Lock)
LED type 0x01 (Caps Lock)
LED type 0x02 (Scroll Lock)

```

The LED example can be directly extended to the other function types. Additional examples, showing EV\_KEY, EV\_REL and EV\_ABS are included in the second part of this document, together with complete forms of the examples in this section.

There is one more bit of information that you need for some devices - the range of values that a particular device can report. Obviously a key can only report whether it is up or down, but devices with absolute axes (such as a joystick or tablet) can report over a basically arbitrary range of values.

We can determine the range of a particular absolute axis using the EVIOCGABS `ioctl`. You have to specify an index, which is the particular axis (for example, ABS\_X or ABS\_THROTTLE) you want to determine the characteristics of.

This `ioctl` provides the information back in a `input_absinfo` structure. This structure is defined as

```
struct input_absinfo {
    uint32_t min_value;
    uint32_t max_value;
    uint32_t fuzz;
    uint32_t flat;
};
```

The `min_value` is the minimum value that this particular axis can return, while the `max_value` is the maximum value that it can return. The `fuzz` element is the range of values that can be considered the same (due to mechanical sensor tolerances, or some other reason), and is zero for most devices. The `flat` is the range of values about the mid-point in the axes that are indicate a zero response (typically, this is the "dead zone" around the null position of a joystick).

Lets look at an example of the EVIOCGABS `ioctl`.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -W`) is provided in the second part of this document.

#### Example 4-6. EVIOCGABS example

```
uint8_t abs_bitmask[ABS_MAX/8 + 1];
struct input_absinfo abs_features;
...
memset(abs_bitmask, 0, sizeof(abs_bitmask));
if (ioctl(fd, EVIOCGBIT(EV_ABS, sizeof(abs_bitmask)), abs_bitmask) < 0) {
    perror("evdev ioctl");
}
for (yalv = 0; yalv < ABS_MAX; yalv++) {
    if (test_bit(yalv, abs_bitmask)) {
        /* this means that the bit is set in the axes list */
        if(ioctl(fd, EVIOCGABS(yalv), &abs_features)) {
            perror("evdev EVIOCGABS ioctl");
        }
        printf("(min: %d, max: %d, flatness: %d, fuzz: %d)\n",
```

```
    abs_features.min_value,  
    abs_features.max_value,  
    abs_features.flat,  
    abs_features.fuzz);  
    }  
}
```

## Getting user input from the device

So far, all of the discussion has been about what the device is capable of (eg whether the device has a trigger button). In this section, we will see how to actually get user input from the device (eg, whether the trigger button has been pressed by the user).

The event interface provides this information using a `read` function call. The result of that `read` is one or more `input_event` structures, and the return value is the number of bytes read.

The `input_event` structure is defined as

```
struct input_event {  
    struct timeval time;  
    unsigned short type;  
    unsigned short code;  
    unsigned int value;  
};
```

The `time` element is a normal `timeval` structure, and contains the time that the event occurred. The `type` element is the type of feature that is being reported (for example, `EV_KEY`, `EV_REL`, `EV_ABS` or `EV_MSC`). The `code` element contains the particular feature that is being reported (for example, `KEY_SPACE`, `KEY_F1`, `REL_WHEEL` or `ABS_X`). The `value` is the value of that particular feature (for example, 0 or 1 for a key, or some other integer value for a relative or absolute axis).

Let's look at an example of the `read` interface.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -W`) is provided in the second part of this document.

### Example 4-7. `read` example

```
    struct input_event ev;  
...  
    while (1)  
    {  
        read(fd, &ev, sizeof(struct input_event));  
  
        printf("Event: time %ld.%06ld, type %d, code %d, value %d\n",  
              ev.time.tv_sec, ev.time.tv_usec, ev.type,  
              ev.code, ev.value);
```

```
}

```

This example shows a busy loop over the particular event device. The `read` call waits for the event (assuming it was opened with `O_NONBLOCK`), and then prints out the various values of the event.

Note that this `read` interface has all the normal characteristics of a character device. This means that you don't need to use a busy loop. You can just wait till your program needs some input from the device, and then perform the `read` call. In addition, if you are interested in the input from a number of devices, you can use the `poll` and `select` functions to wait on a number of open devices at the same time.

For more information on the `read`, `open`, `select` or `poll` functions, refer to the applicable man pages, or any good C programming book.

## Sending information to the device

While input devices don't normally have a lot of data sent to them, there are certain situations where it is useful. For example, a keyboard might have LEDs, or perhaps a buzzer, to indicate mode or error conditions. This section looks at how to send information to an input device.

Sending information to the device is done using the `write` function call on the event interface. This function call takes a `input_device` structure. The `time` element is not used. The `type` is the type of event that is being sent out (for example, `EV_LED`, `EV_SND`, `EV_REP` or `EV_FF`). The `code` is the particular feature that is being manipulated (for example, `LED_NUML`, `LED_CAPSL`, `REP_DELAY` or `SND_BELL`). The `value` is the value to set the particular feature to.

Lets look at an example of the `write` function. This example manipulates the LEDs that are provided on most keyboards and some gamepads.

**About this example:** This example is intentionally a code fragment, and is not complete, nor is it meant to show good programming style.

A complete form of this example (that will compile with `gcc -Wall -w`) is provided in the second part of this document.

### Example 4-8. `write` example

```

struct input_event ev;
...
ev.type = EV_LED;
ev.code = LED_CAPSL;
ev.value = 1;
retval = write(fd, &ev, sizeof(struct input_event));
ev.code = LED_NUML;
retval = write(fd, &ev, sizeof(struct input_event));
ev.code = LED_SCROLLL;
retval = write(fd, &ev, sizeof(struct input_event));

```

## **Modifying key repeat settings**

Use `EVIIOCGREP` and `EVIIOCSREP` `ioctl` functions.

## Chapter 5. Other approaches to Linux HID support

This chapter provides a collection of ideas and thoughts on options other than the hiddev and event interfaces discussed in previous chapters.

### The keyboard interface

To be done.

### The mouse interface

To be done.

### The joystick interface

The joystick interface is provided by the `joydev.o` module, and the outputs appear on the `/dev/input/js0`, `/dev/input/js1` (and so on) device nodes.

More information on programming with the joystick interface is provided in the kernel distribution (see the file `Documentation/input/joystick-api.txt` in a current kernel).

### Force Feedback

Information on programming force feedback effects for joysticks is provided in the kernel distribution (see the file `Documentation/input/ff.txt` in a current kernel).

### A new kernel interface

In addition to all the other interfaces discussed in this document, you could create yet another kernel interface. However, given the flexibility offered by the hiddev and event interfaces, it is difficult to see why this would be needed.

If the interface is intended for wide application, you probably should discuss it on an applicable mailing list (`linux-usb-devel@lists.sourceforge.net` for a USB interface, or `linuxconsole@lists.sourceforge.net` for an input interface, or the main linux-kernel list if neither of those is applicable).



## Chapter 6. HIDDEV examples

**Note:** These complete examples will compile with `gcc -Wall -W`.

If you have updated your linux kernel, and have not updated your header files, you may need to use the `-I` option to pick up the version of `hiddev.h` from your new kernel. For example, if you have the kernel in a directory `/a/b/c`, then you would compile the first example with the command `gcc -Wall -W -I/a/b/c/include example1.c -o example1`. Naturally, you replace `/a/b/c` with the directory you really used.

### Example 6-1. HIDIOCVERSION example

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/hiddev.h>

int main (int argc, char **argv) {

    int fd = -1;
    int version;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s hiddevice - probably /dev/usb/hiddev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("hiddev open");
        exit(1);
    }
}
```

```
/* ioctl() accesses the underlying driver */
ioctl(fd, HIDIOCGVERSION, &version);

/* the HIDIOCGVERSION ioctl() returns an int */
/* so we unpack it and display it */
printf("hiddev driver version is %d.%d.%d\n",
version >> 16, (version >> 8) & 0xff, version & 0xff);

close(fd);

exit(0);
}
```

### Example 6-2. HIDIOCGDEVINFO example

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/hiddev.h>

int main (int argc, char **argv) {

    int fd = -1;
    struct hiddev_devinfo device_info;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s hiddevice - probably /dev/usb/hiddev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("hiddev open");
        exit(1);
    }
}
```

```

/* suck out some device information */
ioctl(fd, HIDIOCGDEVINFO, &device_info);

/* the HIDIOCGDEVINFO ioctl() returns hiddev_devinfo
 * structure - see <linux/hiddev.h>
 * So we work through the various elements, displaying
 * each of them
 */
printf("vendor 0x%04hx product 0x%04hx version 0x%04hx ",
       device_info.vendor, device_info.product, device_info.version);
printf("has %i application%s ", device_info.num_applications,
       (device_info.num_applications==1?"":"s"));
printf("and is on bus: %d devnum: %d ifnum: %d\n",
       device_info.busnum, device_info.devnum, device_info.ifnum);

close(fd);

exit(0);
}

```

### Example 6-3. HIDIOCAPPLICATION example

```

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/hiddev.h>

int main (int argc, char **argv) {

    int fd = -1;
    unsigned int yalv;
    int appl;
    struct hiddev_devinfo device_info;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */

```

```

if (argc != 2) {
    fprintf(stderr, "usage: %s hiddevice - probably /dev/usb/hiddev0\n", argv[0]);
    exit(1);
}
if ((fd = open(argv[1], O_RDONLY)) < 0) {
    perror("hiddev open");
    exit(1);
}

/* suck out some device information */
ioctl(fd, HIDIOCGDEVINFO, &device_info);

/* Now that we have the number of applications (in the
 * device_info.num_applications field),
 * we can retrieve them using the HIDIOCAPPLICATION ioctl()
 * applications are indexed from 0..{num_applications-1}
 */
for (yalv = 0; yalv < device_info.num_applications; yalv++) {
    appl = ioctl(fd, HIDIOCAPPLICATION, yalv);
    if (appl > 0) {
printf("Application %i is 0x%x ", yalv, appl);
/* The magic values come from various usage table specs */
switch ( appl >> 16)
    {
        case 0x01 :
printf("(Generic Desktop Page)\n");
break;
        case 0x0c :
printf("(Consumer Product Page)\n");
break;
        case 0x80 :
printf("(USB Monitor Page)\n");
break;
        case 0x81 :
printf("(USB Enumerated Values Page)\n");
break;
        case 0x82 :
printf("(VESA Virtual Controls Page)\n");
break;
        case 0x83 :
printf("(Reserved Monitor Page)\n");
break;
        case 0x84 :
printf("(Power Device Page)\n");
break;
        case 0x85 :
printf("(Battery System Page)\n");
break;
        case 0x86 :
        case 0x87 :
printf("(Reserved Power Device Page)\n");
break;
        default :
printf("(Unknown page - needs to be added)\n");
    }
    }
}

close(fd);

```

```
    exit(0);  
}
```



## Chapter 7. EVDEV examples

**Note:** These complete examples will compile with `gcc -Wall -W`.

If you have updated your linux kernel, and have not updated your header files, you may need to use the `-I` option to pick up the version of `input.h` from your new kernel. For example, if you have the kernel in a directory `/a/b/c`, then you would compile the first example with the command `gcc -Wall -W -I/a/b/c/include example1.c -o example1`. Naturally, you replace `/a/b/c` with the directory you really used.

### Example 7-1. EVIOCGVERSION example

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/input.h>

int main (int argc, char **argv) {

    int fd = -1;
    int version;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }
}
```

```
/* ioctl() accesses the underlying driver */
if (ioctl(fd, EVIOCGVERSION, &version)) {
    perror("evdev ioctl");
}

/* the EVIOCGVERSION ioctl() returns an int */
/* so we unpack it and display it */
printf("evdev driver version is %d.%d.%d\n",
       version >> 16, (version >> 8) & 0xff, version & 0xff);

close(fd);
exit(0);
}
```

### Example 7-2. EVIOCGID example

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>

#include <linux/input.h>

int main (int argc, char **argv) {

    int fd = -1;
    struct input_devinfo device_info;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
```

```

    perror("evdev open");
    exit(1);
}

/* suck out some device information */
if(ioctl(fd, EVIOCGID, &device_info)) {
    perror("evdev ioctl");
}

/* the EVIOCGID ioctl() returns input_devinfo
 * structure - see <linux/input.h>
 * So we work through the various elements, displaying
 * each of them
 */
printf("vendor 0x%04hx product 0x%04hx version 0x%04hx is on",
device_info.vendor, device_info.product,
device_info.version);
switch ( device_info.bustype)
    {
    case BUS_PCI :
    printf(" a PCI bus\n");
    break;
    case BUS_ISAPNP :
    printf(" a Plug-n-pray ISA bus\n");
    break;
    case BUS_USB :
    printf(" a Universal Serial Bus\n");
    break;
    case BUS_ISA :
    printf(" a legacy ISA bus\n");
    break;
    case BUS_I8042 :
    printf(" an I8042 (or similar) controller\n");
    break;
    case BUS_XTKBD :
    printf(" a IBM XT bus\n");
    break;
    case BUS_RS232 :
    printf(" a RS232 serial bus\n");
    break;
    case BUS_GAMEPORT :
    printf(" a gameport\n");
    break;
    case BUS_PARPORT :
    printf(" a parallel port\n");
    break;
    case BUS_AMIGA :
    printf(" an Amiga unique interface\n");
    break;
    case BUS_ADB :
    printf(" an Apple Desktop Bus\n");
    break;
    case BUS_I2C :
    printf(" a inter-integrated circuit bus\n");
    break;
    default:
    printf(" an unknown bus type: 0x%04hx\n", device_info.bustype);
    }
}

```

```
close(fd);

exit(0);
}
```

### Example 7-3. EVIOCGNAME example

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>

#include <linux/input.h>

int main (int argc, char **argv) {

    int fd = -1;
    char name[256]= "Unknown";

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }

    /* suck out the name information
     * return value is the length of the name, for success
     * or -EFAULT for failure
     */
    if(ioctl(fd, EVIOCGNAME(sizeof(name)), name) < 0) {
        perror("evdev ioctl");
    }
}
```

```

    }

    printf("The device on %s says it's name is %s\n", argv[1], name);

    close(fd);

    exit(0);
}

```

#### Example 7-4. EVIOCGBIT (Event types) example

```

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>

#include <linux/input.h>

/* this macro is used to tell if "bit" is set in "array"
 * it selects a byte from the array, and does a boolean AND
 * operation with a byte that only has the relevant bit set.
 * eg. to check for the 12th bit, we do (array[1] & 1<<4)
 */
#define test_bit(bit, array)    (array[(bit)/8] & (1<<(bit%8)))

int main (int argc, char **argv) {

    int fd = -1;
    uint8_t evtype_bitmask[EV_MAX/8 + 1];
    int yalv;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }

```

```

    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }

    memset(evtype_bitmask, 0, sizeof(evtype_bitmask));
    if (ioctl(fd, EVIOCGBIT(0, EV_MAX), evtype_bitmask) < 0) {
        perror("evdev ioctl");
    }

    printf("Supported event types:\n");

    for (yalv = 0; yalv < EV_MAX; yalv++) {
        if (test_bit(yalv, evtype_bitmask)) {
            /* this means that the bit is set in the event types list */
            printf(" Event type 0x%02x ", yalv);
            switch ( yalv )
            {
                case EV_KEY :
                    printf(" (Keys or Buttons)\n");
                    break;
                case EV_REL :
                    printf(" (Relative Axes)\n");
                    break;
                case EV_ABS :
                    printf(" (Absolute Axes)\n");
                    break;
                case EV_MSC :
                    printf(" (Something miscellaneous)\n");
                    break;
                case EV_LED :
                    printf(" (LEDs)\n");
                    break;
                case EV_SND :
                    printf(" (Sounds)\n");
                    break;
                case EV_REP :
                    printf(" (Repeat)\n");
                    break;
                case EV_FF :
                    printf(" (Force Feedback)\n");
                default:
                    printf(" (Unknown event type: 0x%04hx)\n", yalv);
            }
        }
    }

    close(fd);

    exit(0);
}

```

**Example 7-5. EVIOCGBIT (EV\_LED) example**

```

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>

#include <linux/input.h>

/* this macro is used to tell if "bit" is set in "array"
 * it selects a byte from the array, and does a boolean AND
 * operation with a byte that only has the relevant bit set.
 * eg. to check for the 12th bit, we do (array[1] & 1<<4)
 */
#define test_bit(bit, array)    (array[bit/8] & (1<<(bit%8)))

int main (int argc, char **argv) {

    int fd = -1;
    uint8_t led_bitmask[LED_MAX/8 + 1];
    int yalv;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }

    memset(led_bitmask, 0, sizeof(led_bitmask));
    if (ioctl(fd, EVIOCGBIT(EV_LED, sizeof(led_bitmask)), led_bitmask) < 0) {
        perror("evdev ioctl");
    }
}

```

```

printf("Supported LEDs:\n");

for (yalv = 0; yalv < LED_MAX; yalv++) {
    if (test_bit(yalv, led_bitmask)) {
        /* this means that the bit is set in the LED list */
        printf(" LED type 0x%02x ", yalv);
        switch ( yalv)
        {
            case LED_NUML :
                printf(" (Num Lock)\n");
                break;
            case LED_CAPSL :
                printf(" (Caps Lock)\n");
                break;
            case LED_SCROLLL :
                printf(" (Scroll Lock)\n");
                break;
            case LED_COMPOSE :
                printf(" (Compose)\n");
                break;
            case LED_KANA :
                printf(" (Kana)\n");
                break;
            case LED_SLEEP :
                printf(" (Sleep)\n");
                break;
            case LED_SUSPEND :
                printf(" (Suspend)\n");
                break;
            case LED_MUTE :
                printf(" (Mute)\n");
                break;
            case LED_MISC :
                printf(" (Miscellaneous)\n");
                break;
            default:
                printf(" (Unknown LED type: 0x%04hx)\n", yalv);
        }
    }
}

close(fd);

exit(0);
}

```

**Example 7-6. EVIOCGBIT (EV\_KEY) example**

```

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```

* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>

#include <linux/input.h>

/* this macro is used to tell if "bit" is set in "array"
 * it selects a byte from the array, and does a boolean AND
 * operation with a byte that only has the relevant bit set.
 * eg. to check for the 12th bit, we do (array[1] & 1<<4)
 */
#define test_bit(bit, array)    (array[bit/8] & (1<<(bit%8)))

int main (int argc, char **argv) {

    int fd = -1;
    uint8_t key_bitmask[KEY_MAX/8 + 1];
    int yalv;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }

    memset(key_bitmask, 0, sizeof(key_bitmask));
    if (ioctl(fd, EVIOCGBIT(EV_KEY, sizeof(key_bitmask)), key_bitmask) < 0) {
        perror("evdev ioctl");
    }

    printf("Supported Keys:\n");

    for (yalv = 0; yalv < KEY_MAX; yalv++) {
        if (test_bit(yalv, key_bitmask)) {
            /* this means that the bit is set in the key list */
            printf("  Key 0x%02x ", yalv);
            switch (yalv)
            {
                case KEY_RESERVED : printf(" (Reserved)\n"); break;
                case KEY_ESC : printf(" (Escape)\n"); break;
            }
        }
    }
}

```

```
case KEY_1 : printf(" (1)\n"); break;
case KEY_2 : printf(" (2)\n"); break;
case KEY_3 : printf(" (3)\n"); break;
case KEY_4 : printf(" (4)\n"); break;
case KEY_5 : printf(" (5)\n"); break;
case KEY_6 : printf(" (6)\n"); break;
case KEY_7 : printf(" (7)\n"); break;
case KEY_8 : printf(" (8)\n"); break;
case KEY_9 : printf(" ()\n"); break;
case KEY_0 : printf(" ()\n"); break;
case KEY_MINUS : printf(" (-)\n"); break;
case KEY_EQUAL : printf(" (=)\n"); break;
case KEY_BACKSPACE : printf(" (Backspace)\n"); break;
case KEY_TAB : printf(" (Tab)\n"); break;
case KEY_Q : printf(" (Q)\n"); break;
case KEY_W : printf(" (W)\n"); break;
case KEY_E : printf(" (E)\n"); break;
case KEY_R : printf(" (R)\n"); break;
case KEY_T : printf(" (T)\n"); break;
case KEY_Y : printf(" (Y)\n"); break;
case KEY_U : printf(" (U)\n"); break;
case KEY_I : printf(" (I)\n"); break;
case KEY_O : printf(" (O)\n"); break;
case KEY_P : printf(" (P)\n"); break;
case KEY_LEFTBRACE : printf(" ([)\n"); break;
case KEY_RIGHTBRACE : printf(" (])\n"); break;
case KEY_ENTER : printf(" (Enter)\n"); break;
case KEY_LEFTCTRL : printf(" (LH Control)\n"); break;
case KEY_A : printf(" (A)\n"); break;
case KEY_S : printf(" (S)\n"); break;
case KEY_D : printf(" (D)\n"); break;
case KEY_F : printf(" (F)\n"); break;
case KEY_G : printf(" (G)\n"); break;
case KEY_H : printf(" (H)\n"); break;
case KEY_J : printf(" (J)\n"); break;
case KEY_K : printf(" (K)\n"); break;
case KEY_L : printf(" (L)\n"); break;
case KEY_SEMICOLON : printf(" (;)\n"); break;
case KEY_APOSTROPHE : printf(" (')\n"); break;
case KEY_GRAVE : printf(" (`)\n"); break;
case KEY_LEFTSHIFT : printf(" (LH Shift)\n"); break;
case KEY_BACKSLASH : printf(" (\\)\n"); break;
case KEY_Z : printf(" (Z)\n"); break;
case KEY_X : printf(" (X)\n"); break;
case KEY_C : printf(" (C)\n"); break;
case KEY_V : printf(" (V)\n"); break;
case KEY_B : printf(" (B)\n"); break;
case KEY_N : printf(" (N)\n"); break;
case KEY_M : printf(" (M)\n"); break;
case KEY_COMMA : printf(" (,)\n"); break;
case KEY_DOT : printf(" (.)\n"); break;
case KEY_SLASH : printf(" (/)\n"); break;
case KEY_RIGHTSHIFT : printf(" (RH Shift)\n"); break;
case KEY_KPASTERISK : printf(" (*)\n"); break;
case KEY_LEFTALT : printf(" (LH Alt)\n"); break;
case KEY_SPACE : printf(" (Space)\n"); break;
case KEY_CAPSLOCK : printf(" (CapsLock)\n"); break;
case KEY_F1 : printf(" (F1)\n"); break;
case KEY_F2 : printf(" (F2)\n"); break;
```

```

case KEY_F3 : printf(" (F3)\n"); break;
case KEY_F4 : printf(" (F4)\n"); break;
case KEY_F5 : printf(" (F5)\n"); break;
case KEY_F6 : printf(" (F6)\n"); break;
case KEY_F7 : printf(" (F7)\n"); break;
case KEY_F8 : printf(" (F8)\n"); break;
case KEY_F9 : printf(" (F9)\n"); break;
case KEY_F10 : printf(" (F10)\n"); break;
case KEY_NUMLOCK : printf(" (NumLock)\n"); break;
case KEY_SCROLLLOCK : printf(" (ScrollLock)\n"); break;
case KEY_KP7 : printf(" (KeyPad 7)\n"); break;
case KEY_KP8 : printf(" (KeyPad 8)\n"); break;
case KEY_KP9 : printf(" (Keypad 9)\n"); break;
case KEY_KPMINUS : printf(" (KeyPad Minus)\n"); break;
case KEY_KP4 : printf(" (KeyPad 4)\n"); break;
case KEY_KP5 : printf(" (KeyPad 5)\n"); break;
case KEY_KP6 : printf(" (KeyPad 6)\n"); break;
case KEY_KPPLUS : printf(" (KeyPad Plus)\n"); break;
case KEY_KP1 : printf(" (KeyPad 1)\n"); break;
case KEY_KP2 : printf(" (KeyPad 2)\n"); break;
case KEY_KP3 : printf(" (KeyPad 3)\n"); break;
case KEY_KPDOT : printf(" (KeyPad decimal point)\n"); break;
case KEY_103RD : printf(" (Huh?)\n"); break;
case KEY_F13 : printf(" (F13)\n"); break;
case KEY_102ND : printf(" (Beats me...)\n"); break;
case KEY_F11 : printf(" (F11)\n"); break;
case KEY_F12 : printf(" (F12)\n"); break;
case KEY_F14 : printf(" (F14)\n"); break;
case KEY_F15 : printf(" (F15)\n"); break;
case KEY_F16 : printf(" (F16)\n"); break;
case KEY_F17 : printf(" (F17)\n"); break;
case KEY_F18 : printf(" (F18)\n"); break;
case KEY_F19 : printf(" (F19)\n"); break;
case KEY_F20 : printf(" (F20)\n"); break;
case KEY_KPENTER : printf(" (Keypad Enter)\n"); break;
case KEY_RIGHTCTRL : printf(" (RH Control)\n"); break;
case KEY_KPSLASH : printf(" (KeyPad Forward Slash)\n"); break;
case KEY_SYSRQ : printf(" (System Request)\n"); break;
case KEY_RIGHTALT : printf(" (RH Alternate)\n"); break;
case KEY_LINEFEED : printf(" (Line Feed)\n"); break;
case KEY_HOME : printf(" (Home)\n"); break;
case KEY_UP : printf(" (Up)\n"); break;
case KEY_PAGEUP : printf(" (Page Up)\n"); break;
case KEY_LEFT : printf(" (Left)\n"); break;
case KEY_RIGHT : printf(" (Right)\n"); break;
case KEY_END : printf(" (End)\n"); break;
case KEY_DOWN : printf(" (Down)\n"); break;
case KEY_PAGEDOWN : printf(" (Page Down)\n"); break;
case KEY_INSERT : printf(" (Insert)\n"); break;
case KEY_DELETE : printf(" (Delete)\n"); break;
case KEY_MACRO : printf(" (Macro)\n"); break;
case KEY_MUTE : printf(" (Mute)\n"); break;
case KEY_VOLUMEDOWN : printf(" (Volume Down)\n"); break;
case KEY_VOLUMEUP : printf(" (Volume Up)\n"); break;
case KEY_POWER : printf(" (Power)\n"); break;
case KEY_KPEQUAL : printf(" (KeyPad Equal)\n"); break;
case KEY_KPPLUSMINUS : printf(" (KeyPad +/-)\n"); break;
case KEY_PAUSE : printf(" (Pause)\n"); break;
case KEY_F21 : printf(" (F21)\n"); break;

```

```

case KEY_F22 : printf(" (F22)\n"); break;
case KEY_F23 : printf(" (F23)\n"); break;
case KEY_F24 : printf(" (F24)\n"); break;
case KEY_KPCOMMA : printf(" (KeyPad comma)\n"); break;
case KEY_LEFTMETA : printf(" (LH Meta)\n"); break;
case KEY_RIGHTMETA : printf(" (RH Meta)\n"); break;
case KEY_COMPOSE : printf(" (Compose)\n"); break;
case KEY_STOP : printf(" (Stop)\n"); break;
case KEY_AGAIN : printf(" (Again)\n"); break;
case KEY_PROPS : printf(" (Properties)\n"); break;
case KEY_UNDO : printf(" (Undo)\n"); break;
case KEY_FRONT : printf(" (Front)\n"); break;
case KEY_COPY : printf(" (Copy)\n"); break;
case KEY_OPEN : printf(" (Open)\n"); break;
case KEY_PASTE : printf(" (Paste)\n"); break;
case KEY_FIND : printf(" (Find)\n"); break;
case KEY_CUT : printf(" (Cut)\n"); break;
case KEY_HELP : printf(" (Help)\n"); break;
case KEY_MENU : printf(" (Menu)\n"); break;
case KEY_CALC : printf(" (Calculator)\n"); break;
case KEY_SETUP : printf(" (Setup)\n"); break;
case KEY_SLEEP : printf(" (Sleep)\n"); break;
case KEY_WAKEUP : printf(" (Wakeup)\n"); break;
case KEY_FILE : printf(" (File)\n"); break;
case KEY_SENDFILE : printf(" (Send File)\n"); break;
case KEY_DELETEFILE : printf(" (Delete File)\n"); break;
case KEY_XFER : printf(" (Transfer)\n"); break;
case KEY_PROG1 : printf(" (Program 1)\n"); break;
case KEY_PROG2 : printf(" (Program 2)\n"); break;
case KEY_WWW : printf(" (Web Browser)\n"); break;
case KEY_MSDOS : printf(" (DOS mode)\n"); break;
case KEY_COFFEE : printf(" (Coffee)\n"); break;
case KEY_DIRECTION : printf(" (Direction)\n"); break;
case KEY_CYCLEWINDOWS : printf(" (Window cycle)\n"); break;
case KEY_MAIL : printf(" (Mail)\n"); break;
case KEY_BOOKMARKS : printf(" (Book Marks)\n"); break;
case KEY_COMPUTER : printf(" (Computer)\n"); break;
case KEY_BACK : printf(" (Back)\n"); break;
case KEY_FORWARD : printf(" (Forward)\n"); break;
case KEY_CLOSECD : printf(" (Close CD)\n"); break;
case KEY_EJECTCD : printf(" (Eject CD)\n"); break;
case KEY_EJECTCLOSECD : printf(" (Eject / Close CD)\n"); break;
case KEY_NEXTSONG : printf(" (Next Song)\n"); break;
case KEY_PLAYPAUSE : printf(" (Play and Pause)\n"); break;
case KEY_PREVIOUSSONG : printf(" (Previous Song)\n"); break;
case KEY_STOPCD : printf(" (Stop CD)\n"); break;
case KEY_RECORD : printf(" (Record)\n"); break;
case KEY_REWIND : printf(" (Rewind)\n"); break;
case KEY_PHONE : printf(" (Phone)\n"); break;
case KEY_ISO : printf(" (ISO)\n"); break;
case KEY_CONFIG : printf(" (Config)\n"); break;
case KEY_HOMEPAGE : printf(" (Home)\n"); break;
case KEY_REFRESH : printf(" (Refresh)\n"); break;
case KEY_EXIT : printf(" (Exit)\n"); break;
case KEY_MOVE : printf(" (Move)\n"); break;
case KEY_EDIT : printf(" (Edit)\n"); break;
case KEY_SCROLLUP : printf(" (Scroll Up)\n"); break;
case KEY_SCROLLDOWN : printf(" (Scroll Down)\n"); break;
case KEY_KPLEFTPAREN : printf(" (KeyPad LH parenthesis)\n"); break;

```

```

case KEY_KPRIGHTPAREN : printf(" (KeyPad RH parenthesis)\n"); break;
case KEY_INTL1 : printf(" (Intl 1)\n"); break;
case KEY_INTL2 : printf(" (Intl 2)\n"); break;
case KEY_INTL3 : printf(" (Intl 3)\n"); break;
case KEY_INTL4 : printf(" (Intl 4)\n"); break;
case KEY_INTL5 : printf(" (Intl 5)\n"); break;
case KEY_INTL6 : printf(" (Intl 6)\n"); break;
case KEY_INTL7 : printf(" (Intl 7)\n"); break;
case KEY_INTL8 : printf(" (Intl 8)\n"); break;
case KEY_INTL9 : printf(" (Intl 9)\n"); break;
case KEY_LANG1 : printf(" (Language 1)\n"); break;
case KEY_LANG2 : printf(" (Language 2)\n"); break;
case KEY_LANG3 : printf(" (Language 3)\n"); break;
case KEY_LANG4 : printf(" (Language 4)\n"); break;
case KEY_LANG5 : printf(" (Language 5)\n"); break;
case KEY_LANG6 : printf(" (Language 6)\n"); break;
case KEY_LANG7 : printf(" (Language 7)\n"); break;
case KEY_LANG8 : printf(" (Language 8)\n"); break;
case KEY_LANG9 : printf(" (Language 9)\n"); break;
case KEY_PLAYCD : printf(" (Play CD)\n"); break;
case KEY_PAUSECD : printf(" (Pause CD)\n"); break;
case KEY_PROG3 : printf(" (Program 3)\n"); break;
case KEY_PROG4 : printf(" (Program 4)\n"); break;
case KEY_SUSPEND : printf(" (Suspend)\n"); break;
case KEY_CLOSE : printf(" (Close)\n"); break;
case KEY_UNKNOWN : printf(" (Specifically unknown)\n"); break;
case KEY_BRIGHTNESSDOWN : printf(" (Brightness Down)\n"); break;
case KEY_BRIGHTNESSUP : printf(" (Brightness Up)\n"); break;
case BTN_0 : printf(" (Button 0)\n"); break;
case BTN_1 : printf(" (Button 1)\n"); break;
case BTN_2 : printf(" (Button 2)\n"); break;
case BTN_3 : printf(" (Button 3)\n"); break;
case BTN_4 : printf(" (Button 4)\n"); break;
case BTN_5 : printf(" (Button 5)\n"); break;
case BTN_6 : printf(" (Button 6)\n"); break;
case BTN_7 : printf(" (Button 7)\n"); break;
case BTN_8 : printf(" (Button 8)\n"); break;
case BTN_9 : printf(" (Button 9)\n"); break;
case BTN_LEFT : printf(" (Left Button)\n"); break;
case BTN_RIGHT : printf(" (Right Button)\n"); break;
case BTN_MIDDLE : printf(" (Middle Button)\n"); break;
case BTN_SIDE : printf(" (Side Button)\n"); break;
case BTN_EXTRA : printf(" (Extra Button)\n"); break;
case BTN_FORWARD : printf(" (Forward Button)\n"); break;
case BTN_BACK : printf(" (Back Button)\n"); break;
case BTN_TRIGGER : printf(" (Trigger Button)\n"); break;
case BTN_THUMB : printf(" (Thumb Button)\n"); break;
case BTN_THUMB2 : printf(" (Second Thumb Button)\n"); break;
case BTN_TOP : printf(" (Top Button)\n"); break;
case BTN_TOP2 : printf(" (Second Top Button)\n"); break;
case BTN_PINKIE : printf(" (Pinkie Button)\n"); break;
case BTN_BASE : printf(" (Base Button)\n"); break;
case BTN_BASE2 : printf(" (Second Base Button)\n"); break;
case BTN_BASE3 : printf(" (Third Base Button)\n"); break;
case BTN_BASE4 : printf(" (Fourth Base Button)\n"); break;
case BTN_BASE5 : printf(" (Fifth Base Button)\n"); break;
case BTN_BASE6 : printf(" (Sixth Base Button)\n"); break;
case BTN_DEAD : printf(" (Dead Button)\n"); break;
case BTN_A : printf(" (Button A)\n"); break;

```

```

        case BTN_B : printf(" (Button B)\n"); break;
        case BTN_C : printf(" (Button C)\n"); break;
        case BTN_X : printf(" (Button X)\n"); break;
        case BTN_Y : printf(" (Button Y)\n"); break;
        case BTN_Z : printf(" (Button Z)\n"); break;
        case BTN_TL : printf(" (Thumb Left Button)\n"); break;
        case BTN_TR : printf(" (Thumb Right Button )\n"); break;
        case BTN_TL2 : printf(" (Second Thumb Left Button)\n"); break;
        case BTN_TR2 : printf(" (Second Thumb Right Button )\n"); break;
        case BTN_SELECT : printf(" (Select Button)\n"); break;
        case BTN_MODE : printf(" (Mode Button)\n"); break;
        case BTN_THUMBL : printf(" (Another Left Thumb Button )\n"); break;
        case BTN_THUMBR : printf(" (Another Right Thumb Button )\n"); break;
        case BTN_TOOL_PEN : printf(" (Digitiser Pen Tool)\n"); break;
        case BTN_TOOL_RUBBER : printf(" (Digitiser Rubber Tool)\n"); break;
        case BTN_TOOL_BRUSH : printf(" (Digitiser Brush Tool)\n"); break;
        case BTN_TOOL_PENCIL : printf(" (Digitiser Pencil Tool)\n"); break;
        case BTN_TOOL_AIRBRUSH : printf(" (Digitiser Airbrush Tool)\n"); break;
        case BTN_TOOL_FINGER : printf(" (Digitiser Finger Tool)\n"); break;
        case BTN_TOOL_MOUSE : printf(" (Digitiser Mouse Tool)\n"); break;
        case BTN_TOOL_LENS : printf(" (Digitiser Lens Tool)\n"); break;
        case BTN_TOUCH : printf(" (Digitiser Touch Button )\n"); break;
        case BTN_STYLUS : printf(" (Digitiser Stylus Button )\n"); break;
        case BTN_STYLUS2 : printf(" (Second Digitiser Stylus Button )\n"); break;
        default:
            printf(" (Unknown key)\n");
    }
}

close(fd);

exit(0);
}

```

### Example 7-7. EVIOCGBIT (EV\_REL) example

```

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>

```

```

#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>

#include <linux/input.h>

/* this macro is used to tell if "bit" is set in "array"
 * it selects a byte from the array, and does a boolean AND
 * operation with a byte that only has the relevant bit set.
 * eg. to check for the 12th bit, we do (array[1] & 1<<4)
 */
#define test_bit(bit, array)    (array[bit/8] & (1<<(bit%8)))

int main (int argc, char **argv) {

    int fd = -1;
    uint8_t rel_bitmask[REL_MAX/8 + 1];
    int yalv;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }

    memset(rel_bitmask, 0, sizeof(rel_bitmask));
    if (ioctl(fd, EVIOCGBIT(EV_REL, sizeof(rel_bitmask)), rel_bitmask) < 0) {
        perror("evdev ioctl");
    }

    printf("Supported Relative axes:\n");

    for (yalv = 0; yalv < REL_MAX; yalv++) {
        if (test_bit(yalv, rel_bitmask)) {
            /* this means that the bit is set in the axes list */
            printf("  Relative axis 0x%02x ", yalv);
            switch (yalv)
            {
                case REL_X :
                    printf(" (X Axis)\n");
                    break;
                case REL_Y :
                    printf(" (Y Axis)\n");
                    break;
                case REL_Z :
                    printf(" (Z Axis)\n");
                    break;
                case REL_HWHEEL :
                    printf(" (Horizontal Wheel)\n");
                    break;
                case REL_DIAL :
                    printf(" (Dial)\n");

```

```
        break;
        case REL_WHEEL :
        printf(" (Vertical Wheel)\n");
        break;
        case REL_MISC :
        printf(" (Miscellaneous)\n");
        break;
        default:
        printf(" (Unknown relative feature)\n");
        }
    }
}

close(fd);

exit(0);
}
```

### Example 7-8. EVIOCGBIT (EV\_ABS) example

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>

#include <linux/input.h>

/* this macro is used to tell if "bit" is set in "array"
 * it selects a byte from the array, and does a boolean AND
 * operation with a byte that only has the relevant bit set.
 * eg. to check for the 12th bit, we do (array[1] & 1<<4)
 */
#define test_bit(bit, array)    (array[bit/8] & (1<<(bit%8)))

int main (int argc, char **argv) {
```

```

int fd = -1;
uint8_t abs_bitmask[ABS_MAX/8 + 1];
int yalv;

/* ioctl() requires a file descriptor, so we check we got one, and then open it */
if (argc != 2) {
    fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
    exit(1);
}
if ((fd = open(argv[1], O_RDONLY)) < 0) {
    perror("evdev open");
    exit(1);
}

memset(abs_bitmask, 0, sizeof(abs_bitmask));
if (ioctl(fd, EVIOCGBIT(EV_ABS, sizeof(abs_bitmask)), abs_bitmask) < 0) {
    perror("evdev ioctl");
}

printf("Supported Absolute axes:\n");

for (yalv = 0; yalv < ABS_MAX; yalv++) {
    if (test_bit(yalv, abs_bitmask)) {
        /* this means that the bit is set in the axes list */
        printf("  Absolute axis 0x%02x ", yalv);
        switch (yalv)
        {
            case ABS_X :
                printf(" (X Axis)\n");
                break;
            case ABS_Y :
                printf(" (Y Axis)\n");
                break;
            case ABS_Z :
                printf(" (Z Axis)\n");
                break;
            case ABS_RX :
                printf(" (X Rate Axis)\n");
                break;
            case ABS_RY :
                printf(" (Y Rate Axis)\n");
                break;
            case ABS_RZ :
                printf(" (Z Rate Axis)\n");
                break;
            case ABS_THROTTLE :
                printf(" (Throttle)\n");
                break;
            case ABS_RUDDER :
                printf(" (Rudder)\n");
                break;
            case ABS_WHEEL :
                printf(" (Wheel)\n");
                break;
            case ABS_GAS :
                printf(" (Accelerator)\n");
                break;
            case ABS_BRAKE :

```

```
printf(" (Brake)\n");
break;
    case ABS_HAT0X :
printf(" (Hat zero, x axis)\n");
break;
    case ABS_HAT0Y :
printf(" (Hat zero, y axis)\n");
break;
    case ABS_HAT1X :
printf(" (Hat one, x axis)\n");
break;
    case ABS_HAT1Y :
printf(" (Hat one, y axis)\n");
break;
    case ABS_HAT2X :
printf(" (Hat two, x axis)\n");
break;
    case ABS_HAT2Y :
printf(" (Hat two, y axis)\n");
break;
    case ABS_HAT3X :
printf(" (Hat three, x axis)\n");
break;
    case ABS_HAT3Y :
printf(" (Hat three, y axis)\n");
break;
    case ABS_PRESSURE :
printf(" (Pressure)\n");
break;
    case ABS_DISTANCE :
printf(" (Distance)\n");
break;
    case ABS_TILT_X :
printf(" (Tilt, X axis)\n");
break;
    case ABS_TILT_Y :
printf(" (Tilt, Y axis)\n");
break;
    case ABS_MISC :
printf(" (Miscellaneous)\n");
break;
    default:
printf(" (Unknown absolute feature)\n");
}
}
}
close(fd);
exit(0);
}
```

**Example 7-9. EVIOCGABS example**

```

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>

#include <linux/input.h>

/* this macro is used to tell if "bit" is set in "array"
 * it selects a byte from the array, and does a boolean AND
 * operation with a byte that only has the relevant bit set.
 * eg. to check for the 12th bit, we do (array[1] & 1<<4)
 */
#define test_bit(bit, array)    (array[bit/8] & (1<<(bit%8)))

int main (int argc, char **argv) {

    int fd = -1;
    uint8_t abs_bitmask[ABS_MAX/8 + 1];
    int yalv;
    struct input_absinfo abs_features;

    /* ioctl() requires a file descriptor, so we check we got one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }

    memset(abs_bitmask, 0, sizeof(abs_bitmask));
    if (ioctl(fd, EVIOCGBIT(EV_ABS, sizeof(abs_bitmask)), abs_bitmask) < 0) {
        perror("evdev ioctl");
    }
}

```

```

}

printf("Supported Absolute axes:\n");

for (yalv = 0; yalv < ABS_MAX; yalv++) {
    if (test_bit(yalv, abs_bitmask)) {
        /* this means that the bit is set in the axes list */
        printf(" Absolute axis 0x%02x ", yalv);
        switch ( yalv)
        {
            case ABS_X :
                printf(" (X Axis) ");
                break;
            case ABS_Y :
                printf(" (Y Axis) ");
                break;
            case ABS_Z :
                printf(" (Z Axis) ");
                break;
            case ABS_RX :
                printf(" (X Rate Axis) ");
                break;
            case ABS_RY :
                printf(" (Y Rate Axis) ");
                break;
            case ABS_RZ :
                printf(" (Z Rate Axis) ");
                break;
            case ABS_THROTTLE :
                printf(" (Throttle) ");
                break;
            case ABS_RUDDER :
                printf(" (Rudder) ");
                break;
            case ABS_WHEEL :
                printf(" (Wheel) ");
                break;
            case ABS_GAS :
                printf(" (Accelerator) ");
                break;
            case ABS_BRAKE :
                printf(" (Brake) ");
                break;
            case ABS_HAT0X :
                printf(" (Hat zero, x axis) ");
                break;
            case ABS_HAT0Y :
                printf(" (Hat zero, y axis) ");
                break;
            case ABS_HAT1X :
                printf(" (Hat one, x axis) ");
                break;
            case ABS_HAT1Y :
                printf(" (Hat one, y axis) ");
                break;
            case ABS_HAT2X :
                printf(" (Hat two, x axis) ");
                break;
            case ABS_HAT2Y :

```

```

printf(" (Hat two, y axis) ");
break;
    case ABS_HAT3X :
printf(" (Hat three, x axis) ");
break;
    case ABS_HAT3Y :
printf(" (Hat three, y axis) ");
break;
    case ABS_PRESSURE :
printf(" (Pressure) ");
break;
    case ABS_DISTANCE :
printf(" (Distance) ");
break;
    case ABS_TILT_X :
printf(" (Tilt, X axis) ");
break;
    case ABS_TILT_Y :
printf(" (Tilt, Y axis) ");
break;
    case ABS_MISC :
printf(" (Miscellaneous) ");
break;
    default:
printf(" (Unknown absolute feature) ");
}
if(ioctl(fd, EVIOCGABS(yalv), &abs_features)) {
    perror("evdev EVIOCGABS ioctl");
}
printf("(min: %d, max: %d, flatness: %d, fuzz: %d)\n",
abs_features.min_value,
abs_features.max_value,
abs_features.flat,
abs_features.fuzz);
}
}

close(fd);

exit(0);
}

```

**Example 7-10. read example**

```

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

```

```

*/

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>

#include <linux/input.h>

int main (int argc, char **argv) {

    int fd = -1;          /* the file descriptor for the device */
    int yalv;            /* loop counter */
    size_t read_bytes;  /* how many bytes were read */
    struct input_event ev[64]; /* the events (up to 64 at once) */

    /* read() requires a file descriptor, so we check for one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }

    while (1)
    {
        read_bytes = read(fd, ev, sizeof(struct input_event) * 64);

        if (read_bytes < (int) sizeof(struct input_event)) {
            perror("evtest: short read");
            exit (1);
        }

        for (yalv = 0; yalv < (int) (read_bytes / sizeof(struct input_event)); yalv++)
        {
            printf("Event: time %ld.%06ld, type %d, code %d, value %d\n",
                ev[yalv].time.tv_sec, ev[yalv].time.tv_usec, ev[yalv].type,
                ev[yalv].code, ev[yalv].value);
        }

        close(fd);

        exit(0);
    }
}

```

**Example 7-11. write (EV\_LED) example**

```

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <asm/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <errno.h>

#include <linux/input.h>

int main (int argc, char **argv) {

    int fd = -1;          /* the file descriptor for the device */
    int retval = 0;      /* bytes written in write() */
    struct input_event ev; /* the event */

    /* read() requires a file descriptor, so we check for one, and then open it */
    if (argc != 2) {
        fprintf(stderr, "usage: %s event-device - probably /dev/input/evdev0\n", argv[0]);
        exit(1);
    }
    if ((fd = open(argv[1], O_WRONLY)) < 0) {
        perror("evdev open");
        exit(1);
    }

    /* we turn off all the LEDs to start */
    ev.type = EV_LED;
    ev.code = LED_CAPSL;
    ev.value = 0;
    retval = write(fd, &ev, sizeof(struct input_event));
    ev.code = LED_NUML;
    retval = write(fd, &ev, sizeof(struct input_event));
    ev.code = LED_SCROLLL;
    retval = write(fd, &ev, sizeof(struct input_event));
}

```

```
while (1)
{
    ev.code = LED_CAPSL;
    ev.value = 1;
    retval = write(fd, &ev, sizeof(struct input_event));
    usleep(200000);

    ev.value = 0;
    retval = write(fd, &ev, sizeof(struct input_event));
    ev.code = LED_NUML;
    ev.value = 1;
    write(fd, &ev, sizeof(struct input_event));
    usleep(200000);

    ev.value = 0;
    retval = write(fd, &ev, sizeof(struct input_event));
    ev.code = LED_SCROLLL;
    ev.value = 1;
    write(fd, &ev, sizeof(struct input_event));
    usleep(200000);

    ev.value = 0;
    retval = write(fd, &ev, sizeof(struct input_event));
}

close(fd);

exit(0);
}
```

## Chapter 8. Contributors

I am indebted to all the developers who have worked on Linux USB and Input subsystems. I'd particularly like to thank Vojtech Pavlik and Paul Stewart, who are primarily responsible for the excellent HID support in current Linux kernels.



## Chapter 9. Availability and licensing

### Obtaining updates and translations

You can get the latest version of this Guide from the Linux USB Project<sup>1</sup>.

You can get the SGML source by emailing the author, or from the Linux USB Project Sourceforge page<sup>2</sup>.

There are no known translations of this document at this time.

### License

This document is Copyright (C) Brad Hards (1999-2002).

This document may be distributed only subject to the terms and conditions set forth in the Linux Documentation Project License<sup>3</sup>.

### Notes

1. <http://www.linux-usb.org/>
2. <http://sourceforge.net/projects/linux-usb/>
3. <http://www.tldp.org/COPYRIGHT.html>



## Chapter 10. Corrections

Please send comments on this document to the author, preferably by E-Mail (<bradh-usb@frogmouth.net>), including the version number: (\$Revision: 0.1 \$).

